**Лабораторна робота №16 Модифікація власних програм з додаванням нових функцій з передачею параметрів за різними механізмами**

**Мета:**. навчитися писати програми **на мові С++** в **консольному режимі** з використанням функцій та файлів, навчитися модифікувати власні програми**.**.

**Хід роботи**

1. Повторити Лекції 13-14.
2. Запустити середовище програмування С++ .
3. Записати програму, що виконує завдання з пп.4. В першому рядку програми записати

*// Група № Прізвище Номер ЛР*

вказавши номер своєї групи та своє прізвище.

Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних.

1. **Завдання**: Модифікуємо програму з лабораторної роботи №15, додавши можливість оброблення додаткової інформації. Використати напрацювання п.3 лабораторної роботи №10: включити інформацію про осіб, які гіпотетично можуть проживати в будівлях, дані про які зафіксовані в ЛР№15. Додати відповідні позиції меню, які визначають функції:
2. Запис інформації про особу, де заповнюється в циклі масив структур, контролюючи, щоб кількість осіб не перевищила розмірності масиву.

Структура повинна містити поля (див. лабораторну роботу №10):

* код (порядковий номер),
* прізвище та ініціали,
* стать (1 – жіноча, 2 чоловіча),
* телефон,
* адреса,
* професія.

Коли заповнення завершене, інформація записується в бінарний файл. Структура файлу: у перших двох байтах розміщується значення типу int, що визначає кількість зроблених у файл записів; далі без пропусків розміщаються записи про осіб.

1. Закрити файл для запису.
2. Відкрити файл для читання.
3. Виведення всієї інформації по особах
4. Виведення інформації по окремій особі. Запитує номер потрібного запису (код - порядковий номер) , зчитує потрібну інформацію та виводить на консоль. Дії шодо файлу визначте самостійно.

Зауваження. При реалізації дозапису інформації до файлу потрібно буде спочатку визначити / прочитати інформацію про кількість раніш сформованих записів для продовження правильного встановлення кодів.

1. Результати у вигляді текстового файлу надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**ОП+АМ <Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, 21-01Ivanov.cpp.

**Тему в заголовку листа записати**

**ОП+АМ <Номер групи>-><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ОП+АМ-Запитання-<Номер групи>-<Прізвище >**.

**Строк відсилки ЛР для ІПЗ-21**

**ІПЗ-22 13.05.2022**

***Контрольні запитання для самоперевірки***.

1. Які класи використовують для роботи з файловими потоками?

2. Які режими доступу можна використовувати під час роботи з файлами? Наведіть приклади.

3. Який метод слугує для відкриття файлу? Наведіть приклади.

4. Які операції доступні для роботи з файлами? Які функції призначені для виконання цих операцій?

5. Які методи дозволяють визначити кінець файлу під час читання інформації? У чому відмінність цих методів? Наведіть приклади.

6. Як можна вважати змінні стандартних типів даних із текстових файлів?

7. Чи можна зчитувати з текстових файлів змінні типів даних користувача?

8. Які функції призначені довільного читання інформації з файлу? Наведіть приклади.

9. Назвіть особливості бінарних файлів. У чому переваги використання таких файлів?

10. За допомогою яких функцій можна записувати/зчитувати інформацію у двійкові файли?

11. Як вважати змінні стандартних типів даних із двійкового файлу?

12. Які особливості потрібно враховувати при читанні типів даних користувача з двійкових файлів?

**ТЕОРЕТИЧНА ЧАСТИНА**

**Фізичний і логічний файли**

Поняття файлу можна розглядати з двох точок зору. З одного боку, файл - це іменована область на зовнішньому носії інформації, що містить довільні дані. Файл у такому розумінні називають *фізичним* файлом, тобто таким, що існує фізично на матеріальному носії інформації. З іншого боку, файл — це одна із структур даних, що використовується у програмуванні. У такому розумінні файл називають *логічним,* тобто таким, що існує в певній програмі як абстракція.

Файл як фізичний об'єкт є послідовністю байтів. Фізичний файл характеризується іменем, що його ідентифікує. Розмір файлу може бути довільним і обмежується лише ємністю пристроїв зовнішньої пам'яті.

Файл як логічний об'єкт є послідовністю значень певного типу, тобто він складається з однотипних компонентів.

**Бінарні файли**

Бінарні файли зберігають інформацію у вигляді, як вона представлена у пам'яті комп'ютера, і тому незручні для людини. Заглянувши в такий файл, неможливо зрозуміти, що в ньому записано; його не можна створювати чи виправляти вручну - у якомусь текстовому редакторі - тощо. Однак усі ці незручності компенсуються швидкістю роботи з даними.

Крім того, текстові файли відносяться до структур послідовного доступу, а бінарні – прямого. Це означає, що у будь-який момент часу можна звернутися до будь-якого, а не лише до поточного елементу

**Робота з двійковими файлами**

Вся інформація зберігається у комп'ютері у вигляді 0 та 1, тобто у двійковому вигляді. Двійкові файли відрізняються від текстових лише методами роботи з ними. Наприклад, якщо ми записуємо в текстовий файл цифру «4», вона записується як символ, і її зберігання потрібен один байт. Відповідно і розмір файлу дорівнюватиме одному байту. Текстовий файл, що містить запис: "145687", матиме розмір шість байт.

Якщо ж записати ціле число 145687 в двійковий файл, то він матиме розмір чотири байти, так як саме стільки необхідно для зберігання даних типу int. Тобто двійкові файли компактніші і в деяких випадках більш зручні для обробки.

**Запис стандартних типів даних у двійкові файли**

Щоб відкрити двійковий файл, необхідно встановити режим доступу ios::binary (у деяких компіляторах C++ - ios::bin).

Для створення вихідного файлу створюють об'єкт:

**ofstream outBinFile("out.bin", ios::out | ios::binary);**

**/\* *створення об'єкта класу* ofstream out. bin \*/**

**if(!out\_f == 1) *//стандартна перевірка***

Запис даних відбувається за допомогою методу write(), який має два параметри: перший - покажчик на початок (адреса початку) даних, що записуються, другий - кількість байтів, що записуються. При цьому вказівник необхідно явно перетворити на тип char.

*Приклад 1.*Записати в двійковий файл змінні різного типу:

**#include <iostream>**

**#include <fstream>**

**#include <Windows.h>**

**using namespace std;**

**int main() {**

**system("color F0");**

**ofstream out("test.bin",ios::out | ios::binary); */\*Створення об'єкта класу* of stream *та спроба зв'язати його з файлом* test. bin *у режимі запису двійкового файлу \*/***

**int a = 145687; *//Оголошення цілої змінної* а**

**out.write ((char\*) &a, sizeof(a)); */\*запис у файл змінної* а *як потоку байтів, тобто запис у файл внутрішнього уявлення цілої змінної* а \*/**

**float x = 123.25; // *оголошення дійсної змінної* х**

**out.write ((char\*) &x, sizeof (x)) ; */\*запис у файл змінної* х *як потоку байтів, тобто. запис у файл внутрішнього уявлення цілої змінної х \*/***

***/\* Визначення символьної змінної* с *та ініціалізація її символом g* \*/**

**char c = 'g';**

**out.write((char\*)&c, sizeof(c));**

***//запис символу* g *у файл***

**out.close();**

**return 0;}**

Якщо відкрити вміст файлу test .bin текстовим редактором, то він матиме вигляд:

9ЂцBg

а розмір файлу становитиме 9 байт.

**Читання стандартних типів даних із двійкових файлів**

Щоб відкрити існуючий двійковий файл для читання, потрібно створити об'єкт:

ifstream inpBinFile("inp.bin", ios::in | ios::binary);

/\* *використовуємо диз'юнкцію прапорів, що вказує на те, що файл відкривається на читання в двійковому вигляді\*/*

if (! inpBinFile)

Для читання даних використовуємо функцію read(), що має аналогічні функції write() параметри.

#include…

using namespace std;

int main(){

ifstream inpBinFile("test.bin", ios::in | ios:: binary); /*\*відкриваємо файл на читання у двійковому вигляді\*/*

int а; float х; char c ;

inpBinFile.read((char\*)&a, sizeof(a)); *//читаємо цілу чисельну змінну*

inpBinFile.read((char\*)&x, sizeof(x)); *//читаємо дійсну змінну*

inpBinFile.read((char\*)&c, sizeof(c)); *//читаємо символьну змінну*

inpBinFile.close();

cout …

return 0;

}

Результат роботи програми:

а = 145687 х = 123.25 с = g

Зверніть увагу, що при використанні функції write та read не відбувається жодного перетворення інформації. У файл записується та зчитується внутрішнє подання даних.

**Запис і читання типів даних користувача в двійкові файли**

На відміну від текстових файлів, робота з типами даних користувача з використанням двійкових файлів нічим не відрізняється від стандартних типів даних. Аналогічно використовуються методи write() та read(). Програмістові тільки залишається вказати адресу записуваної ділянки пам'яті і кількість байтів, що записуються, врахувавши при цьому, що ніякого перетворення даних не відбувається, записується і зчитується тільки внутрішнє подання інформації.

*приклад 3.*Написати програму, яка записує відомості про групу туристів у двійковий файл.

struct Anketa {/\**структурного типу даних* Anketa *\*/*

char name;

int age;}

const int n=10;

…

int main(){

fstream BinFile("ankety.bin", ios::in | ios::out | ios::binary);

Anketa Gruppa [n];

for … // В циклі заповнюємо i-ту анкету

{…

BinFile.write((char\*)&Gruppa[i], sizeof(Anketa)); }

BinFile.close();

return 0;}

Також під час роботи з двійковими файлами можуть використовуватися методи seekg(), tellg(), seekp(), tellp().

**Файловий вказівник**

Кожен **клас файлового вводу/виводу** містить файловий вказівник, який використовується для відстеження поточної позиції читання/запису даних в файлі. Будь-який запис в файл або читання вмісту файлу відбувається в поточному розташуванні файлового вказівника. За замовчуванням, при відкритті файлу для читання або запису, файловий вказівник знаходиться на самому початку цього файлу. Однак, якщо файл відкривається в режимі додавання, то файловий вказівник переміщається в кінець файлу, щоб користувач мав можливість додати дані в файл, а не перезаписати його.

**Рандомний доступ до файлів за допомогою функцій seekg() і seekp()**

До цього моменту ми здійснювали послідовний доступ до файлів, тобто виконували читання/запис файлу по порядку. Проте, ми можемо виконати і **довільний (рандомний) доступ** до файлу (тобто переміщатися по файлу, як захочемо). Це може бути корисно, коли файл має великий вміст, а нам потрібен всього лише невеликий конкретний запис з цього всього. Замість послідовного доступу (коли ми переходимо до потрібного запису починаючи з самого початку файлу), ми можемо здійснити безпосередній доступ до цього запису.

Рандомний доступ до файлу здійснюється шляхом маніпулювання файловим вказівником за допомогою **функції seekg()** (закінчення **“g”** = “***g***et”, тобто “отримати/дістати”) — для вводу, і **функції seekp()** (закінчення ***“p”*** = “***p***ut” (тобто “покласти/помістити”) — для виводу.

Функції **seekg()** і **seekp()** приймають наступні **два параметри**:

**перший параметр** — це зміщення на яке слід перемістити файловий вказівник (вимірюється в байтах);

**другий параметр** — це флаг [**ios**](https://acode.com.ua/urok-217-funktsional-klasiv-ostream-i-ios-formatuvannya-vyvodu/), який позначає місце, від якого слід відштовхуватися при виконанні зміщення.

**Флаги ios**, які приймають функції seekg() і seekp() в якості другого параметра:

**beg** — зміщення відносно початку файлу (за замовчуванням);

**cur** — зміщення відносно поточного розташування файлового вказівника;

**end** — зміщення відносно кінця файлу.

Для визначення позиції, в якій знаходиться покажчик потоку в даний момент, використовуються функції:

**in.tellg()** та **in.tellp()** — відповідно при читанні та записі файла,

де іn — це ім’я потоку, створеного операторами ifstream() або ofstream().

**Метод tellg ()**

Іноді потрібно отримувати інформацію про те, скільки вже прочитано. У цьому допоможе метод **tellg**():

cout << "зчитано байт:" << file.tellg ();

Він повертає значення типу **int**, яке показує скільки вже пройдено в байтах. Його можна використовувати в парі з методом **seekg**(), щоб отримувати розмір файлу:

// стаємо в кінець файлу

file.seekg (0, ios\_base :: end);

// Отримуємо поточну позицію

cout << "Розмір файлу (в байтах):" << file.tellg ();